Simultaneous animation of multiple independent parameters

Wagner L. Truppel

October 28, 2019

The problem

Supoose we have a function $f(p_1, p_2, \ldots, p_N)$ that depends on N parameters $\{p_1, p_2, \ldots, p_N\}$ and we want to animate that function by animating the individual parameters, independently, from their initial values to their final values, with different animation properties for each parameter.[1]

More concretely, we'd like to animate each parameter p_i subjected to the following animation properties:

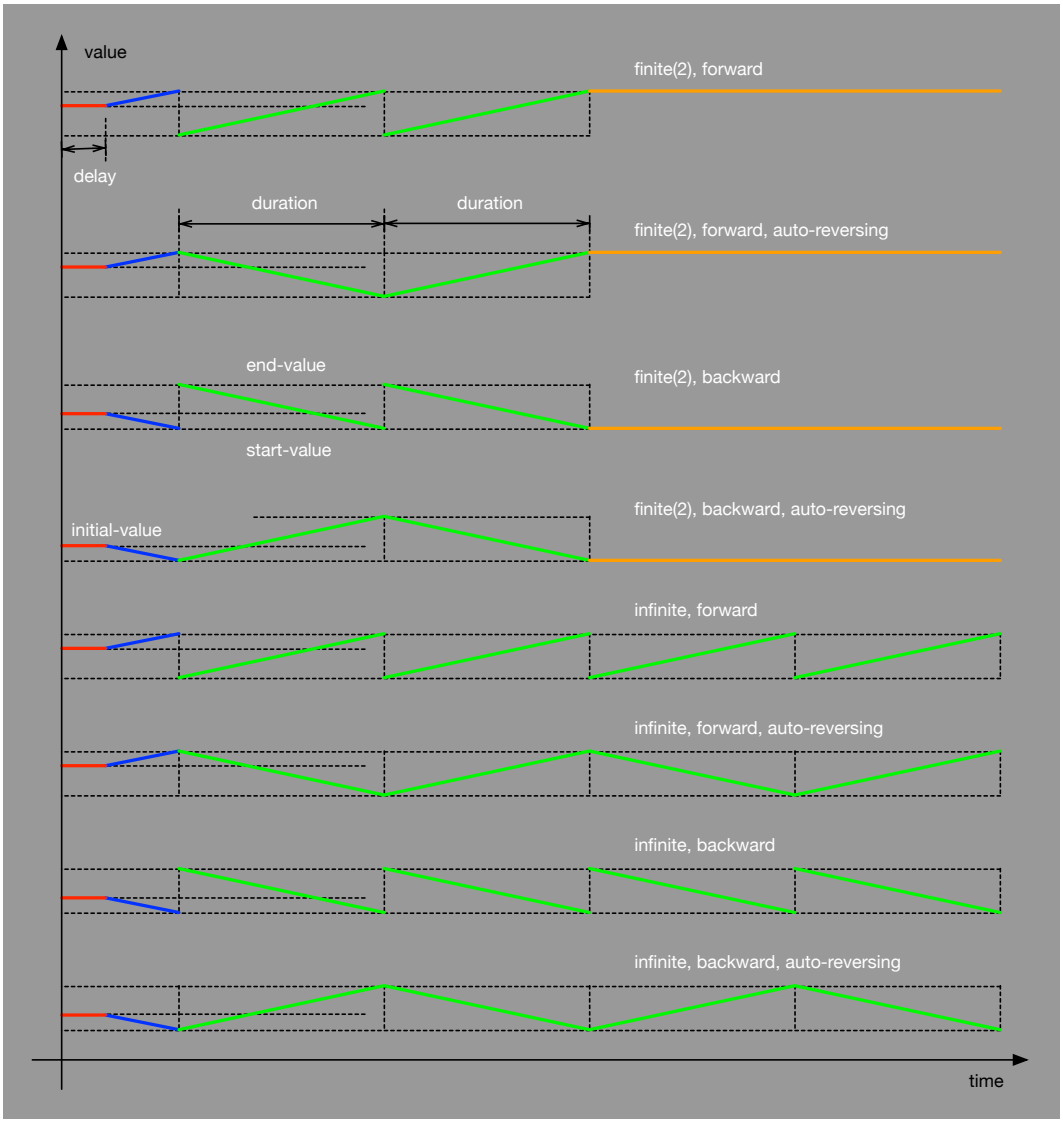
- `startValue` ($\texttt{sv}_i$), `endValue` ($\texttt{ev}_i$): the parameter's end-point values;
- `initialValue` ($\texttt{iv}_i$): the parameter's initial value;
- `delay` (δ_i): the delay before starting the parameter animation;
- `duration` (Δt_i): the duration of the parameter animation;
- `initialDirection` ($\texttt{idir}_i$): the initial direction of the parameter animation, whether from `startValue` to `endValue` (`forward`) or from `endValue` to `startValue` (`backward`);
- `autoReverses` ($\texttt{arevs}_i$): whether or not the animation automatically reverses its direction upon reaching a parameter end-point;
- `numberOfRepetitions` ($\texttt{nreps}_i$): the number of complete traversals of the range `startValue` $\rightarrow$ `endValue` or of the range `endValue` $\rightarrow$ `startValue`. An animation that has a finite number $n \in \mathcal{N}$ of repetitions has an $\texttt{nreps}_i$ value of `finite`(n) while one that repeats itself indefinitely has an $\texttt{nreps}_i$ value of `infinite`.

[1] For example, we could have a shape that is built from a function such as

$$r(\phi) = \frac{a+b}{2} + \frac{a-b}{2}\cos(n\,\phi + c + d\cos\phi + e\sin\phi) + f\cos\phi + g\sin\phi + h$$

where $\phi \in [\,0, 2\pi)$, $n \in \mathcal{N}^+$, $0 \leq a < b \leq 1$, $(c, d, e, f, g, h) \in \mathcal{R}$, with $(c, d, e) \in [\,0, 2\pi)$ and $(f, g, h) \in [-1, 1]$. Here, the interesting animatable parameters are n, a, b, c, d, e, f, g, and h, while ϕ is not animatable, but could be.

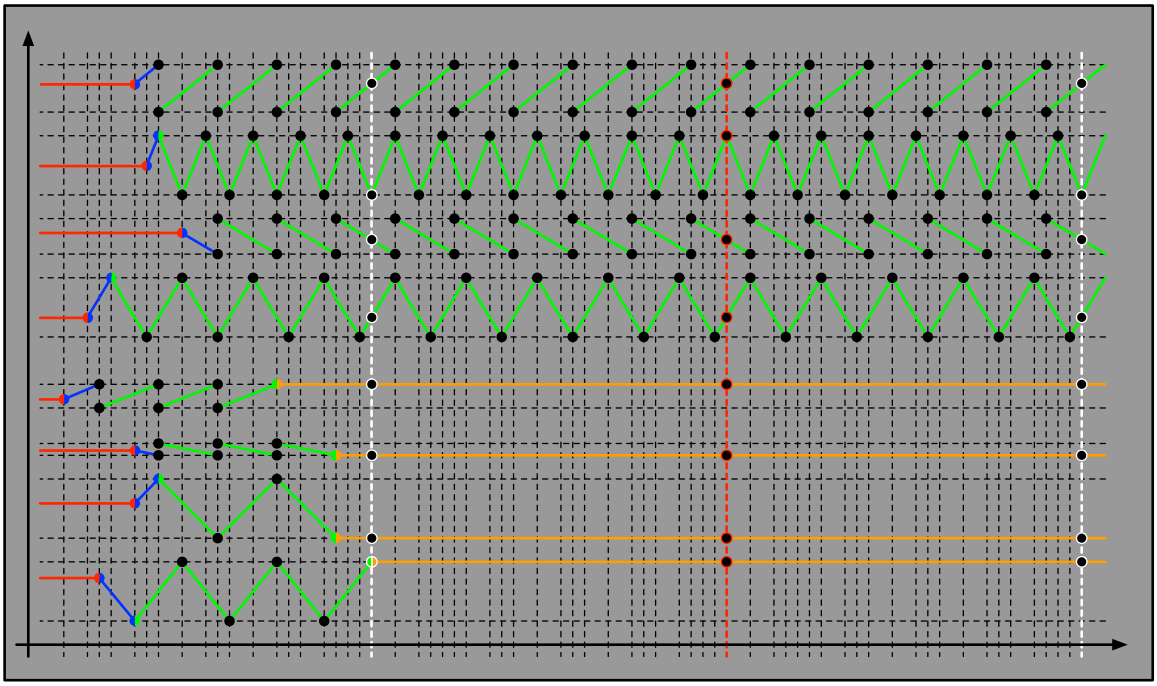
These animation properties are illustrated below:



Note how the parameter's value stays at `initialValue` until the animation starts and, if the animation has a finite number of repetitions, stays at its final value after all animation runs are completed.

The problem is that computing the function for every instant of time is an expensive proposition, especially if it needs to be done at 60 frames per second. We'd like, therefore, to find appropriate *key-frames* and compute the function only at those points in time, allowing

the animation framework to interpolate between the computed key-frames (which it can do efficiently). The appropriate key-frames are, naturally, those computed at points in time (a) when a parameter's animation starts and (b) when it reaches either end-point. For an example, refer to the image below.



For the illustrated example, the longest-running finite animation (the bottom-most one in the figure) is completed at the time indicated by the left-most white dashed line. By that time, all finite animations are completed but some of the indefinitely-repeating ones are in mid-flight.

We can split the entire animation process into two phases. The first phase animates all parameters until all the finite animations are completed (that is, until that first white line). The second phase then animates all parameters indefinitely from their values at the first white line until they repeat themselves as a set (that is, until the second white line). The key-frames computed for this second phase can be cached by the animation framework so we only need to compute them once. Of course, there won't be a second phase if there are no indefinitely-repeating animations.

A subtlety that requires special consideration is the fact that it isn't sufficient for the parameters to repeat their *values*. Their *directions* also need to match. For instance, in the first, third, and fourth animations from the top, in the figure, the parameter values *and their directions* at the red line are the same as their values at the first white line but the

second animation has its *direction flipped* at the moment indicated by the red line. We'll come back to this issue below.

A key-frame is essentially the collection of the values of all parameters at a given moment in time. If we set a coordinate axis for time, then each parameter animation starts at its `delay`, δ_i, then proceeds for a certain period of time dt_i (which depends on the parameter's initial value $\texttt{iv}_i$ and on the animation's initial direction $\texttt{idir}_i$) until the animation reaches one of the parameter's end-points. From that point on, the next end-point is an amount of time away equal to the parameter animation's `duration`, Δt_i. The value of dt_i is easily computed by interpolation,

$$dt_i = \frac{\Delta t_i}{(\texttt{ev}_i - \texttt{sv}_i)} \begin{cases} (\texttt{ev}_i - \texttt{iv}_i)\,, & \text{if } \texttt{idir}_i = \texttt{forward}\,, \\ (\texttt{iv}_i - \texttt{sv}_i)\,, & \text{if } \texttt{idir}_i = \texttt{backward}\,, \end{cases}$$

an expression that is valid regardless of whether $\texttt{ev}_i > \texttt{sv}_i$ or $\texttt{ev}_i < \texttt{sv}_i$. The moment of time when the i-th parameter animation reaches an end-point after completing n repetitions is then given by

$$t_i(n) = \delta_i + dt_i + n\,\Delta t_i\,,$$

where $n \geq 0$ and, for finite animations, $n \leq \texttt{nreps}_i$. This expression lets us compute the moment of time when a finite animation completes, its `completion-time` ($\texttt{ct}_i$),

$$\texttt{ct}_i = t_i(\texttt{nreps}_i) = \delta_i + dt_i + \texttt{nreps}_i\,\Delta t_i\,.$$

Therefore, the longest-running finite animation, and last to complete, will complete at the moment of time given by

$$\texttt{ct} = \max_i(\texttt{ct}_i) = \max_i\,\left(\delta_i + dt_i + \texttt{nreps}_i\,\Delta t_i\right)\,.$$

This is the moment when phase 1 ends and phase 2, if existing, starts. We can now compute the phase 1 key-frame times:

- For all animations:
$$t_i(n) = \delta_i$$
- For finite animations:
$$t_i(n) = \delta_i + dt_i + n\,\Delta t_i\,, \quad 0 \leq n \leq \texttt{nreps}_i$$
- For indefinitely-repeating animations:
$$t_i(n) = \delta_i + dt_i + n\,\Delta t_i\,, \quad n \geq 0\,,$$
with the condition that $t_i(n) \leq \texttt{ct}$, which is what limits n for a given value of i.

The complete set of key-frame times is then the set-union of these two collections of time values, sorted in ascending order. We now need to compute the parameter values for all of these instants of time. A careful examination of the example illustrated previously gives:

$$p_i(t) = \begin{cases} \texttt{iv}_i\,, & \text{if } t \le \delta_i\,, \\ \text{if } \delta_i < t \le \delta_i + dt_i: & \\ \quad \texttt{iv}_i + \dfrac{(\texttt{ev}_i - \texttt{iv}_i)(t-\delta_i)}{dt_i}\,, & \text{if } \texttt{dir}_i(t) = \texttt{forward}\,, \\ \quad \texttt{iv}_i - \dfrac{(\texttt{iv}_i - \texttt{sv}_i)(t-\delta_i)}{dt_i}\,, & \text{if } \texttt{dir}_i(t) = \texttt{backward}\,, \\ \text{if } \delta_i + dt_i < t < \texttt{ct}_i: & \\ \quad \texttt{sv}_i + \dfrac{(\texttt{ev}_i - \texttt{sv}_i)}{\Delta t_i}\,[\,(t-\delta_i-dt_i)\,\%\,\Delta t_i\,]\,, & \text{if } \texttt{dir}_i(t) = \texttt{forward}\,, \\ \quad \texttt{ev}_i - \dfrac{(\texttt{ev}_i - \texttt{sv}_i)}{\Delta t_i}\,[\,(t-\delta_i-dt_i)\,\%\,\Delta t_i\,]\,, & \text{if } \texttt{dir}_i(t) = \texttt{backward}\,, \\ \text{if } t \ge \texttt{ct}_i: & \\ \quad \texttt{ev}_i\,, & \text{if } \texttt{dir}_i(\texttt{ct}_i) = \texttt{forward}\,, \\ \quad \texttt{sv}_i\,, & \text{if } \texttt{dir}_i(\texttt{ct}_i) = \texttt{backward}\,, \end{cases}$$

where $\texttt{ct}_i$ is considered to be infinite for indefinitely-repeating animations and $\texttt{dir}_i(t)$ is given by:

$$\texttt{dir}_i(t) = \begin{cases} \texttt{idir}_i & \text{if } \texttt{arevs}_i = \texttt{false}\,, \\ \text{same as } \texttt{idir}_i & \text{if } \texttt{arevs}_i = \texttt{true} \text{ and } \lfloor \dfrac{(t-\delta_i-dt_i)}{\Delta t_i} \rfloor \text{ is } \texttt{odd}\,, \\ \text{opposite of } \texttt{idir}_i & \text{if } \texttt{arevs}_i = \texttt{true} \text{ and } \lfloor \dfrac{(t-\delta_i-dt_i)}{\Delta t_i} \rfloor \text{ is } \texttt{even}\,. \end{cases}$$

Up until this point, time could be continuous but we'll need it to be discretized in what follows. We should choose a unit that is short enough to result in smooth animations but long enough to avoid having a large number of key-frames. If the goal is to have animations that are smooth at 60 Hz then we should choose 1/60 of a second as the unit of time. Regardless, we'll now consider time to be discrete.

For phase 2, assuming it happens (it won't happen unless at least one parameter is supposed to be animated indefinitely), we start with the parameters at their values for $t =$

`ct`. They will repeat themselves individually at their respective durations but, collectively, they'll only repeat their values again after a span of time (measured from $t = \texttt{ct}$) equal to the *least common multiple* of their durations, and this is the reason we need time to be discrete (so we *can*, in fact, calculate a least common multiple).

It isn't that simple, however, for parameters whose animations auto-reverse their directions. Since their directions flip at successive animation runs, they'll repeat their parameter values *and their directions* only after an *even* number of runs. Thus, the equivalent of a completion time for indefinitely-repeating animations is equal to

$$\texttt{ct}' = \texttt{ct} + \text{lcm}(\,\{\Delta t'_i\}\,)$$

where

$$\Delta t'_i = \begin{cases} \Delta t_i\,, & \text{if } \texttt{arevs}_i = \texttt{false}\,, \\ 2\Delta t_i\,, & \text{if } \texttt{arevs}_i = \texttt{true}\,. \end{cases}$$

The key-frame times for phase 2, then, are computed solely from the indefinitely-animated parameters, using

$$t_i(n) = \delta_i + dt_i + n\,\Delta t_i\,, \quad n \geq 0\,,$$

with the condition that $t_i(n) \leq \texttt{ct}'$. Once we have these times, we can use the same expressions written earlier to compute the values of the animatable parameters at those times, and we have a complete solution to the problem.

The only remaining consideration is how to extend the results above to animations with curves that aren't linear. That is something I haven't approached yet.

■